2007-05

Finding Logically Consistent Resource-Deception Plans for Defense in Cyberspace

Rowe, Neil C.

Monterey, California. Naval Postgraduate School

Finding Logically Consistent Resource-Deception Plans for Defense in Cyberspace

Neil C. Rowe
U.S. Naval Postgraduate School, Monterey, California 93943 USA
ncrowe at nps.edu

Abstract

We explore a new approach to defense of computer systems, deliberately deceiving attackers as to resource availability. This can be more effective than outright denial of access because it encourages an attacker to waste time continuing their attack. But effective deceptions must be consistent to convince an adversary. We are exploring automated methods for maintaining logical consistency by tracking assertions made so far with associated causal and other indirect implications. We have built a deception planner that takes as input as sequence of operating-system commands and finds the possible consistent deceptions as per our logical constraints, and rates the deceptions using several criteria. In a test on a generic planning model of rootkit installation, it found 72 of 558 possible deceptions were acceptable and rated them.

This paper appeared in the Third International Symposium on Security in Networks and Distributed Systems, Niagra Falls, Ontario, Canada, pp. 563-568, May 2007.

1. Introduction

Passwords, keys, and other access control are insufficient to protect computer systems from malicious attack today, as is witnessed by the many attacks that circumvent them. One promising secondary line of defense is deliberate deception to impede attacks since deceptions are classic and often very-successful military tactics [1]. Honeypots (systems designed to be attacked to learn about methods) are the most common defensive deception in cyberspace [2]. Deception can encourage attackers to go away by making them think your computer system cannot be exploited [3]. But here we consider a different deception, encouraging the attacker to think that their difficulties are only temporary, staying online and wasting their time. This has been termed a "tarpit" since it tries to get the attacker "stuck", as in the deliberately delaying responses of the LaBrea tool [4]. This would be valuable in time-critical attacks on important systems as during information warfare [5], to permit time to implement countermeasures such as IP address blocking. Encouraging the attacker to stay online also provides more data about them. While there are ethical concerns about deception, most ethical theories permit deception to prevent a significantly greater harm, and destruction of the software of a computer system by an exploit is serious.

As an example, suppose a user logs in to a system, connects to the Internet, and downloads an executable file. This is not suspicious. However, suppose they next try to download a larger executable in which the signature of a Trojan horse is apparent. Rather than refuse, which will just encourage them to find an easier attack target, we could lie that the network is not working, or pretend to download the file but not actually do so, or download it but modify characters in it to make it useless. But these deceptions will be unconvincing since the user successfully downloaded the previous file. While the network and the file-transfer utility could have stopped working in the meantime, it is unlikely and suspicious to an attacker. It would be more convincing if we lie that the file is too large to transfer.

So we need to plan deceptions carefully. We would like to find logically consistent deceptions in a systematic way whenever a user starts to appear malicious as judged by an intrusion-detection system [6]. Often more than one deception will be possible and we must rank them. Resource denials often make especially good deceptions since it is often hard to contradict the reasons given by a computer system.

2. Background

Deception is a common social phenomenon. It would be impossible for societies to function without it in many areas such as law, politics, business, entertainment, and psychology [7]. Attackers of computer systems deceive about their identities and software tools, so it would seem fair to use deception to defend systems too. Deception is a surprising defense for computers, and can be difficult for attackers to perceive.

Many models of deception have been proposed, including those in psychology [8] and counterintelligence [9]. One good military taxonomy is that of concealment, camouflage, disinformation, lies, displays, ruses, demonstrations, feints, and insight [1]. Excuses are a versatile strategy for refusing to do things [10], and so are good tactics for deception. But none of this work treats deception as a problem in formal logic although people tend to treat deception as either present or not [11].

Vrij identifies seven characteristics of "good liars" [12]. Computer systems intrinsically satisfy the properties of thinking quickly, having a good memory, not experiencing fear or guilt, and being good at pretending. Eloquence is not required nor necessary for computers. So being well prepared and being original are the key challenges for computerized lying. This suggests that for a computer to deceive, it must have a good plan with some surprises in it.

Deception is essential to honeypots since attackers do not want data collected about them. Attackers can go to great lengths to detect honeypots [13]. Thus honeypots should try to convince attackers [14] by concealing their monitoring processes and populating their memory with fake data suggesting normal usage. Fake data can include passwords to other machines, credit-card numbers, and other things which we can confirm are being used, or it can serve counterintelligence purposes, providing data for spies.

Deception by a computer system may hurt legitimate users of that system who are doing something unusual. We have analyzed this problem elsewhere using cost-benefit analysis [15] and provided guidelines, so we do not discuss it here.

3. Logical modeling of the attacker

Consider a computer system under attack by a relatively alert attacker who is trying to achieve particular goals. This could be an insider or an organization-sponsored outsider, or even a sophisticated amateur. At the same time, the tactics we propose here of deceptive denial of requested resources are unexpected events for an attack script, and thus will easily stop most automated attacks. Our approach has similarities to the work of [16] on maintaining confidentiality of databases by tracking logical inferences, but here we generalize well beyond data confidentiality.

3.1 Resources, facets, and parameters

We wish to protect computer systems by giving consistent false excuses for resource denial to a suspicious user. Consistency requires remembering all resources used during a session, including those used indirectly, and their observed status. For instance, when a user successfully downloads a file from the network, this says that the network, the local file system, and the remote file system are working, at least then and for that kind of file. Inconsistency in resource availability will at least make the user suspicious and suspicion may cause behavior like retaliation that is difficult to predict since people often have emotional reactions to being deceived [11]. A network could stop working during a session, but that is unlikely in such a short time period. So we propose:

The Statelessness Assumption: The resources of a computer system remain constant in their availability status through a session of a user.

The resources associated with commands to an operating system are the "material" of attacks [17]:

- The directories and files of the computer;
- Peripheral devices to which the computer is attached;
- Networks to which the computer is attached;
- Other sites accessible by the networks;
- The executables for the commands run by the operating system;
- Status markers such as "logged-in" and "administrator privileges"; and
- People. We will ignore this category because they are hard to control, though "social engineering" does use them.

Resources associated with commands can be identified in several ways. Many commands give the resource as an argument, like the site "server23" for the command "ftp server23", though some arguments are just parameters like numbers and strings. Resources can persist over a sequence of commands even when not again mentioned, as the "network" resource in commands to a file-transfer (ftp) utility; and resources can be "released" by particular commands, like an edited file after exiting the editor. In addition, the executable implementing a command is a resource for it. It is useful to distinguish constant resources like computer systems and

networks from created resources like downloaded files and administrator privileges. Resource denial is more convincing for created resources because they are less tested. However, some attacks have few created resources to exploit.

For each resource, we propose six facets of its status, each with an associated predicate:

- Existence, exists(X): Whether the resource X exists (at least where you are looking);
- Authorization, authorized(X): Whether the user is authorized to use the resource X (as by passwords and access control);
- Readiness, initialized(X,A): Whether the preparation of resource X is sufficient for the action A to be done;
- Operability, working(X): Whether the resource X is functionally sound;
- Compatibility, compatible(X,Y): Whether the two resources X and Y are mutually compatible (for instance, a text editor is incompatible with an image);
- Moderation, moderate(X,A): Whether the action's demands on a resource are within acceptable limits.

Deceptions will be most effective on the last four facets since it appears to an attacker that they are closer to success, encouraging them to further waste time. Still, deceptions on the first two can provide variety, and they sound like traditional access control. The "moderation" facet has ranges of suitable parameter values which can be inferred. For instance, successful download of a one-megabyte file suggests that a smaller file could also be transferred but not necessarily a larger file. Some parameters are:

- Files: size, authorization level
- Peripheral devices: size, bandwidth
- Networks: size, bandwidth
- Sites: load, number of users
- Passwords: length, number of times used

As an example, suppose Bob downloads a file "foobar.doc" of size 50,000 bytes from "remotesite" to "homesite" across network "localnet" via the FTP file-transfer utility on homesite, at a time when localnet has five simultaneous users already. Normal completion of the action says that file systems on remotesite and homesite exist, are authorized access by Bob, are initialized for access, and are working. Similarly it says the network localnet exists, is authorized use by Bob, is initialized for file transfers, is working, and is compatible with remotesite and homesite. It also says executable ftp exists on homesite, Bob is authorized to use it, it is initialized, it is working, it is compatible with the file system on homesite, it is compatible with localnet, and it is compatible the file system on remotesite. Finally, the file system on homesite can hold files of 50,000 bytes, localnet can transfer files of 50,000 bytes, and localnet can handle six simultaneous users.

3.2 Resource-status inference rules

Rules can infer the status of one resource based on the status of another, and these can be used to check for inconsistencies. Our approach is similar to that of debugging by finding inconsistencies in source code and its behavior [18], as when a program attempts to dereference an uninitialized variable.

When a command is successfully reported as accomplished, the issuer assumes that each facet of each input resource is valid. But outputs are not necessarily working, initialized, or compatible since other actions may need to be done first, as with a downloaded compressed file. We can formulate this in formal logic without time arguments since we assume consistency over time:

$$\forall X \forall A \forall Y [(exists(X) \wedge authorized(X) \wedge$$
$$initialized(X, A) \wedge working(X) \wedge$$
$$compatible(X, Y) \wedge moderate(X, A))$$
$$\leftarrow (done(A) \wedge input(A, X)) \wedge$$
$$(input(A, Y) \vee output(A, Y)))]$$
$$\forall X \forall A[(exists(X) \wedge authorized(X))$$
$$\leftarrow (done(A) \wedge output(A, X))]$$

When a command fails, usually one facet is blamed in an error message. If more than one can be blamed, usually only the last one in our above list is indicated. So if your attempt to download fails with the message “Network file limit exceeded”, you can infer that the site exists, the file exists, you are authorized to access them, transfer is initialized properly, and transfer is otherwise working. In general:

$$\forall X[exists(X) \leftarrow authorized(X)]$$
$$\forall X \forall A[authorized(X) \leftarrow initialized(X, A)]$$
$$\forall X \forall A[initialized(X, A) \leftarrow working(X)]$$
$$\forall X \forall Y[working(X) \leftarrow compatible(X, Y)]$$
$$\forall X \forall Y[compatible(X, Y) \leftarrow compatible(Y, X)]$$
$$\forall X \forall A[compatible(X, A) \leftarrow moderate(X, A)]$$

Note that the contrapositives are useful too, so if something does not exist, no other facets apply to it.

Part-whole inferences are also useful. For instance, if a file-transfer buffer is not working, then neither is a utility that relies upon it. In general, negative status inherits upwards from parts to wholes, and positive status inherits downwards from wholes to parts. Using second-order logic for each facet F:

$$\forall F \forall X \forall Y[F(X) \leftarrow (part\_of(X, Y) \wedge F(Y))$$
$$\forall F \forall X \forall Y[\sim F(X) \leftarrow$$
$$(part\_of(Y, X) \wedge \sim F(Y))$$

Resources can be required indirectly by a command. For instance, a file transfer requires access to the network of the remote site although its name is not given in the file-transfer command. In general:

$$\forall A \forall X \exists Y[input(A, X) \leftarrow$$
$$(input(A, Y) \wedge needs(Y, X))]$$

Attacker knowledge can also be expressed as rules. For instance, if the file-transfer utility FTP always seems to break when the attacker tries to transfer a 10000 byte file, the attacker would learn:

$$\forall X \forall A \exists S[\neg working(executable(ftp))$$
$$\leftarrow (a\_kind\_of(A, ftp\_command) \wedge$$
$$input(A, X) \wedge size(X, S) \wedge (S > 10000))]$$

Inferences are also possible from the context of the deception. Attacking a computer system often requires a multistep plan, and many of these plans are known. We should check deceptions against nearby suspicious actions in the inferred plan so that unwanted causal inferences cannot be made by the attacker. In a rootkit installation plan for instance, there is usually only one notably suspicious action like a buffer overflow. If we were to deceive just after it by saying the network was down, we would be creating an unwanted causal inference that the buffer overflow caused the statement that the network was down. The best explanation of such a causal link between two unrelated entities is deliberate deception by the computer system, what we are trying to conceal. On the other hand, a claimed memory-protection violation is a possible consequence of the buffer overflow and deception would not seem as likely an explanation. So deceptive statements need to appear causally related only to innocent actions.

Besides a priori suspicious actions like buffer overflows, we should avoid causal connections of deceptions to critically important actions in the attack plan. For example, if we issue a protection violation on testing the rootkit during a rootkit installation plan, the attacker will be quite suspicious because this is essential to achieving their main goal of having a working rootkit; it would look like the system is desperate to do the last thing that it can to foil the plan. If on the other hand we claim a protection violation on the decompressing of a rootkit, that is a lesser subgoal and the error message does not seem as suspicious. Hence we propose:

The Suspicion-Causation Constraint: A deception cannot be used within a step of a suspicious attacker action unless it is clearly causally related to the suspicious action by an everyday understanding of computer software. A suspicious action is one whose a priori suspiciousness exceeds a threshold or at which a major attack goal is achieved.

3.3 Deception tactics

Once we have chosen to deceive an attacker on some resource and facet, we have several tactics we can use, based on the 24 general defensive deception methods for cyberspace in [19]:

- To deceive on existence, issue an error message that the resource does not exist. For instance, after a file transfer to the current site, say that the operating system cannot find the transferred file.
- To deceive on authorization, either say that the user is not authorized to use the resource, or ask the user to provide

credentials (a password or key) which we then reject.

- To deceive on readiness, issue an error message immediately on attempting to use the resource. For instance, after an executable is downloaded, say "Protection violation at location 6349573" in trying to use it.
- To deceive on operability, either stop in the middle of apparent execution with an error message, or appear to never terminate execution, or create unusable result resources (which is also a deception on their readiness).
- To deceive on compatibility, issue an error message citing two resources. For instance, if the user is doing a network file transfer, claim that the remote site is not recognizing our transfer protocol.
- To deceive on moderation, claim that a limit has been exceeded.

A deception should usually be chosen to appear to prevent completion of an attacker's inferred goals. For instance, if we wish to prevent an attacker from doing a rootkit download, we pretend the network is down so they cannot get it. Ensuring this requires some reasoning about attack plans, but attacker goals are not highly varied (install a rootkit, make a site unusable, steal information, etc.) and many attacks are well documented [20].

The Statelessness Assumption means any deception must also be backwards-consistent with the resource facets reported so far. This means that as an attack proceeds, there are fewer options for deception (the deceiver is "boxed into a corner"). This suggests deception should generally be done as early as possible once a user is inferred to be an attacker. Alternatively, we may choose to violate the Statelessness Assumption on occasion, particularly in a slow session.

4. Quantifying deceptions

Even after logical criteria rule out inconsistent and suspicious deceptions, there still may be many possible deceptions during a sequence of user commands. So it is helpful to rate them. A reasonable criterion is the a priori likelihood of the deception event occurring. The factors we use are:

- A priori likelihood of lack of problems with the associated facet of availability;
- A priori likelihood of lack of problems with the associated resource;
- Whether the resource is created (which makes resource denial more plausible); and
- Suspiciousness of the associated command (it increases deception desirability).

These can be estimated from statistics on similar computer systems. We ignore here the vigilance of the user, and assume that detection of the deception is a monotonic function of its rating on the factors. Note that the degree to which the user might be nonmalicious is a constant factor over all deception methods and thus can be ignored in comparing them. However, we should not be deceiving at all if this probability is significant.

If we are willing to violate the Statelessness Assumption, the a priori probability of this inconsistency represents a fifth factor. It can be estimated by the simplest queuing model as $1-e^{-\lambda t}$ where t is time and λ is the expected number of times that system condition D involved in the deception would change during a unit time period (its service rate).

5. Experiments with a deception planner

As a demonstration of these ideas, we wrote a Gnu Prolog program that computes good resource deceptions for sequences of commands issued to a Unix operating system. Resources must be specifically enumerated for each command for use by the deception planner, to which is added a default resource of the executable for each command. For a test, command sequences were created by running the rootkit-installation planner of [21]. This is a hierarchical planner in Prolog that associates preconditions, postconditions, goal-recommendation condition, and random changes with each command. Its rootkit-installation model had 110 possible actions. We used Naïve-Bayes odds multiplication to rate logically-acceptable opportunities using the first four factors of section 4; suspiciousness of the associated command multiplied the odds by 1.9, and whether the resource was created multiplied the odds by 1.6. Probability estimates for some of these came from the survey of users in [22].

Here are example deceptions proposed by our program on one 41-step rootkit-installation plan generated by the planner. Weights are the ratings. (The highest-rated should not always be chosen, since some surprise is helpful to the effectiveness of deceptions.) Only the first opportunity to use the deception in a command sequence for a resource is listed; all subsequent attempts to use the same resource will fail similarly. Each deception of these could be worded in several ways for the user, so there are additional implementation choices.

For command download(rootkit, hackerhome, patsy): Lie by saying the user is not authorized for file(rootkit). [weight 0.285]

For command download(rootkit, hackerhome, patsy): Ask the user for a password to enable use of file(rootkit), then claim it is invalid. [weight 0.285]

For command download(rootkit, hackerhome, patsy): Abort execution midway with error message concerning file(rootkit). [weight 0.150]

For command download(rootkit, ,hackerhome, patsy): Falsely claim that executable(download) and file(rootkit) are incompatible. [weight 0.150]

For command overflow(buffer, port80, patsy): Falsely claim that buffer(buffer_of(patsy)) and port(port80) are incompatible. [weight 0.090]

For command download(rootkit, hackerhome, patsy): Falsely claim that file(rootkit) exceeds maximum size by having value 100000. [weight 0.077]

The full analysis of this attack plan found 72 deception opportunities in the above format for the 41-step plan. This was for 9 deception tactics and 62 resource-action pairs total, giving 558 resource-action-deception triples, so only 12.9% of the possible deception opportunities were logically valid by our rules (and only 7.3% of those had a rating of 0.05). This percentage will decrease further for plans of more steps because of the difficulty of consistency in longer plans. Thus our rules are significantly selective in proposing deception opportunities.

6. Conclusions

We have examined a new approach to defending computer systems, that of having them deliberately deceive attackers about availability of system resources to waste their time. Using a theory with inference rules, we have built a deception planner that finds a small set of logically consistent deceptions for an attack plan. Consistency is important because counterintelligence training emphasizes the importance of finding discrepancies to detect deception.

Acknowledgements

This work was supported by the National Science Foundation under the Cyber Trust Program. The views expressed are those of the author and do not represent policy of the U.S. Government.

References

[1] Dunnigan, J., and Nofi, A., *Victory and deceit, second edition: deception and trickery in war*. San Jose, CA: Writers Club Press, 2001.

[2] The Honeynet Project, Know your enemy: learning about security threats, 2nd edition. Boston: Addison-Wesley, 2004.

[3] Rowe, N., Duong, B., and Custy, E., Fake honeypots: a defensive tactic for cyberspace. 7th IEEE Workshop on Information Assurance, West Point, NY, June 2006, pp. 223-230.

[4] Anonymous, Web pages retrieved February 18, 2007 from labrea.sourceforge.net.

[5] Hutchinson, W., and Warren, M., Information warfare: corporate attack and defense in a digital world, London: Butterworth-Heinemann, 2001.

[6] Endorf, C., Schultz, E., and Mellander, J., *Intrusion detection and prevention*. Emeryville, CA: McGraw-Hill Osborne, 2004.

[7] Nyberg, D., *The varnished truth: truth telling and deceiving in ordinary life*. Chicago: University of Chicago Press, 1993.

[8] Heuer, R. J., Cognitive factors in deception and counterdeception. In *Strategic Military Deception*, ed. Daniel, D. C., Herbig, K. L., New York: Pergamon, 1982, pp. 31-69.

[9] Whaley, B., Towards a general theory of deception. *Journal of Strategic Studies*, Vol. 5, No. 1, pp. 179-193, March 1982.

[10] Snyder, C. R., Higgins, R. L., and Stucky, R. J., *Excuses: masquerades in search of grace*. New York: Wiley, 1983.

[11] Ford, C. V., *Lies! Lies!! Lies!!! The psychology of deceit*. Washington, DC: American Psychiatric Press, 1996.

[12] Vrij, A., *Detecting lies and deceit: the psychology of lying and the implications for professional practice*. Chichester, UK: Wiley, 2000.

[13] Holz, T., and Raynal, F., Detecting honeypots and other suspicious environments, Proc. 6th SMC Information Assurance Workshop, West Point, NY, pp. 29-36, June 2005.

[14] Cohen, F., and Koike, D., Leading attackers through attack graphs with deceptions. *Computers and Security*, Vol. 22, no. 5, pp. 402-411, 2003.

[15] Rowe, N., Planning cost-effective deceptive resource denial in defense to cyber-attacks. Proc. 2nd International Conference on Information Warfare, Monterey, CA, March 2007.

[16] Spalka, A., Formal semantics of confidentiality in multilevel logic databases. Proc. New Security Paradigms Workshop, Little Neck, RI, pp. 64-73, 1994.

[17] Templeton, S., and Levitt, K., A requires / provides model for computer attacks. Proc. of the New Security Paradigms Workshop,

Cork, Ireland, September 2000.
[18] Engler, D., Chen, D., Hallem, S., Chou, A., Chelf, B., Bugs as deviant behavior: a general approach to inferring errors in systems code. Proc. 18th ACM Symposium on Operating System Principles, Banff, Alberta, Canada, 57-72, 2001.
[19] Rowe, N., and Rothstein, H., Two taxonomies of deception for attacks on information systems. *Journal of Information Warfare*, Vol. 3, No. 2 (July), 2004, 27-39.
[20] Chirillo, J., *Hack attacks revealed*, Wiley, New York, 2002.
[21] Rowe, N., Counterplanning deceptions to foil cyber-attack plans. Proc. 2003 IEEE-SMC Workshop on Information Assurance, West Point, NY, June 2003, 203-211.
[22] Rowe, N., Designing good deceptions in defense of information systems. Computer Security Applications Conference, Tucson, AZ, 418-427, December 2004.